UNIVERSIDAD DE LA RIOJA

TRABAJO FIN DE ESTUDIOS

Título

ProbleMath: un recurso educativo abierto de problemas matemáticos

Autor/es

ALEJANDRO MAHÍLLO CAZORLA

Director/es

ARTURO JAIME ELIZONDO y ÓSCAR CIAURRI RAMÍREZ

Facultad

Facultad de Ciencia y Tecnología

Titulación

Grado en Ingeniería Informática

Departamento

MATEMÁTICAS Y COMPUTACIÓN

Curso académico

2019-20

publicaciones.unirioja.es
E-mail: publicaciones@unirioja.es

UNIVERSIDAD DE LA RIOJA

Facultad de Ciencia y Tecnología

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

ProbleMath: un recurso educativo abierto de problemas matemáticos

Realizado por:

Alejandro Mahillo Cazorla

Tutelado por:

Arturo Jaime Elizondo y Óscar Ciaurri Ramírez

Logroño, Junio de 2020

Resumen

En este proyecto se ha desarrollado desde cero una aplicación web de código abierto para el almacenamiento, clasificación y búsqueda de problemas resueltos, escritos en LaTeX, permitiendo la consulta de los documentos compilados desde el navegador y la elaboración de hojas de problemas en formato PDF. La aplicación es de acceso público con excepción de la introducción de problemas que exige el inicio de sesión. El proyecto se ha desarrollado de forma autónoma y fue propuesto por el profesor Óscar Ciaurri Ramírez para un conjunto de problemas del grado de matemáticas. El producto obtenido ha tratado de cumplir sus requisitos y necesidades más importantes. Aunque se hizo un estudio previo de alternativas de código abierto adaptables al proyecto, fueron descartadas por la complejidad que entrañaban.

Las principales dificultades que han surgido son de tipo tecnológico. No ha sido sencillo, por ejemplo, confeccionar documentos LaTeX a partir de otros, o gestionar la visualización compilada de documentos LaTeX en el navegador. También se ha desplegado la solución en un servidor de la universidad, superando sus controles de seguridad.

El producto es una solución en tres capas, utiliza MySQL, la API/REST se ha desarrollado en Python con el framework Flask y la interfaz web en PHP. También se han utilizado funcionalidades para la gestión de LaTeX como conversores de imágenes (pdftoppm), visualizadores (MathJax y TikzJax), o un compilador de LaTeX (TexLive). El producto se ha publicado en GitHub bajo una licencia GNU GPLv3.

Abstract

In this project an open source web application has been developed from scratch for the storage, classification and search of solved problems, written in LaTeX, allowing the consultation of compiled documents from the browser and the elaboration of problem sheets in PDF format. The application is publicly accessible with the exception of the problem upload that requires login. The project has been developed autonomously and was proposed by Professor Óscar Ciaurri Ramírez for a set of problems in the mathematics grade. The product obtained has tried to meet his most important requirements and needs. Although a previous study of open source alternatives adaptable to the project was made, they were discarded because of the complexity involved.

The main difficulties that have arisen are of a technological nature. It has not been easy, for example, to build LaTeX documents from others, or to manage the compiled display of LaTeX documents in the browser. The solution has also been deployed on a server at the university, exceeding its security controls.

The product is a three-tier solution, using MySQL, the API/REST has been developed in Python with the Flask framework and the web interface in PHP. Functionalities for the management of LaTeX have also been used, such as image converters (pdftoppm), viewers (MathJax and TikzJax), or a LaTeX compiler (TexLive). The product has been published in GitHub under a GNU GPLv3 license.

Índice

1. Introducción

En este apartado justificamos la elección del proyecto desarrollado en este trabajo fin de grado (TFG), los motivos que llevaron a realizarlo, por qué lo propuso el cliente, resumimos los objetivos principales y presentamos cómo se organiza la memoria.

1.1. Contexto

El TFG es una propuesta del profesor Óscar Ciaurri Ramírez y surge de su necesidad de reorganizar y hacer públicos unos materiales que había ido desarrollando a lo largo de varios años. El material es una colección amplia de enunciados y soluciones de problemas de matemáticas escritos en LaTeX. La idea inicial era poner el material a disposición del público en general y en particular de los estudiantes de grado de matemáticas. También se pretendía facilitar las búsquedas y la generación automática de hojas de problemas en formato legible.

Actualmente el perfil de matemático es muy demandado por empresas de diferentes tipos. Los egresados adquieren una gran capacitación en resolución de problemas. Pero, para lograrlo, es preciso asimilar la base aportada por la teoría aplicándola posteriormente con criterio a situaciones específicas. Para lograr esta competencia es fundamental tener acceso a conjuntos de problemas apropiados. Normalmente el profesor hace una selección de problemas que ofrece a sus estudiantes. En el caso que nos ocupa, la generación de una hoja de problemas es una tarea laboriosa, sobre todo la selección de problemas adecuados y variados. Con el confinamiento al que nos hemos visto sometidos en la crisis de la COVID-19, todos nos hemos visto forzados a seguir un modelo telemático, al que no estamos acostumbrados en una universidad presencial. La solución desarrollada en este TFG puede ser muy útil para mejorar situaciones semejantes, para potenciar el autoaprendizaje y para apoyar la tarea de los profesores en la composición de hojas de problemas.

1.2. Motivación

El trabajo a realizar en este proyecto ofrece varios puntos destacables e interesantes. En primer lugar está la temática abordada. Al realizar este trabajo voy a poder aportar desde el mundo de la informática algo al mundo de las matemáticas, uniendo de alguna forma las dos disciplinas que he estado estudiando los últimos años. También utiliza el sistema LaTeX para composición de textos, muy utilizado en la publicación de artículos, especialmente del área de las matemáticas.

En segundo lugar el proyecto es una petición de un cliente real, profesor del grado de matemáticas. Esto supone que se va a poner a prueba la capacidad de comunicación con él en la captura de requisitos, verificación de versiones, etc.

En tercer lugar, es muy importante desplegar una versión funcional del producto realizado en servidores de la universidad. En la universidad existen otros precedentes de productos para el aprendizaje desarrollados por estudiantes en sus TFG. Un caso concreto es AplicaciónBD, utilizado en las prácticas de consultas SQL. Este paso de implantación no lo había realizado en ninguna de las asignaturas del grado.

Por último, el producto está orientado a su utilización por profesores y estudiantes, con lo que el esfuerzo realizado puede ser útil como herramienta de autoaprendizaje y repositorio para la publicación de problemas de otros profesores.

Por todo lo anterior, la propuesta me pareció muy interesante y decidí aceptarla. Además, lógicamente, desarrollaría la solución en alguna tecnología, a decidir durante el proyecto, que me ayudaría a reforzar lo aprendido durante la titulación. El proyecto tenía un reto adicional, la necesidad de realizar el proyecto por mi cuenta, sin el respaldo de un equipo de profesionales, como ocurre en los realizados en empresa.

Esta memoria la organizamos de la siguiente manera. Comenzaremos explicando los antecedentes, es decir cómo se gestionaban las hojas de problemas antes del proyecto. A continuación se describen los objetivos del TFG y un resumen del alcance, incluyendo la licencia del producto realizado. Le sigue el capítulo de gestión, que se trata de una gestión post-mortem. En el apartado de análisis y diseño se explica el estudio sobre adaptación de soluciones existentes, que se hizo inicialmente y las decisiones que le siguieron. Después se describen brevemente los principales requisitos funcionales abordados, la arquitectura y la estructura de datos manejados en la solución. El apartado sobre la implementación realizada, explica qué herramientas y tecnologías se han utilizado, los problemas principales encontrados y sus soluciones. La sección de implantación, explica el despliegue en servidores de producción y el proceso seguido para garantizar la seguridad del entorno.

La memoria finaliza con una sección de conclusiones donde se destacan los puntos relevantes de la experiencia y se trata de extraer alguna lección aprendida.

2. Antecedentes

El proyecto desarrollado en este TFG surge de la necesidad particular de un profesor y es un esfuerzo individual. Aparte de proponer el problema y sus necesidades, el profesor no impuso ninguna tecnología ni metodologías de desarrollo. Se buscaba una solución web de código abierto y accesible por cualquier persona interesada en los contenidos a ofrecer. Los interesados en el proyecto se reducen al propio profesor que propone el problema.

En esta sección vamos a explicar cómo se hacía la gestión manual de problemas (ejercicios) y sus soluciones, antes de proponer este proyecto. También comentaremos las posibilidades de mejora que se observaban.

LaTeX. Como los problemas y soluciones que maneja nuestro cliente están escritos en LaTeX, comenzamos explicando qué es y cómo funciona. LaTeX es un sistema de composición de textos que consigue resultados de muy buena calidad tipográfica y es muy utilizado en la escritura de artículos científicos, en especial en el área de matemáticas por sus facilidades para la escritura de fórmulas. Los documentos se escriben en texto plano combinado con instrucciones, como por ejemplo \begin{itemize}...\end{itemize} que crea listas de viñetas. También incluye instrucciones para insertar imágenes desde archivos (jpg, gif…) o para generar dibujos. Podemos ver un ejemplo en el anexo A (Código LaTeX). Un documento LaTex tiene extensión *.tex* y es necesario compilarlo para obtener el documento legible con las tipografías aplicadas, normalmente un PDF. El compilador incluye también las imágenes en el resultado y genera los dibujos. El anexo B (LaTeX compilado) contiene el resultado de compilar el documento LaTeX del anexo A.

Almacenamiento de problemas. Había una carpeta para cada tema. Dentro de cada una de ellas había una carpeta por cada problema. En la carpeta de un problema se situaba su fichero *.tex* y su fichero PDF.

Búsqueda de problemas. Se visitaba la carpeta del tema deseado y se iban consultando los problemas moviéndose a cada carpeta de problema y abriendo el documento PDF.

Confección de una hoja de problemas. Se creaba primero un nuevo documento *.tex* para la nueva hoja de problemas. Para cada problema seleccionado en una búsqueda, como la que acabamos de explicar, se abría su documento *.tex,* se copiaba el código y se pegaba en el documento de la hoja de problemas. Se terminaba el proceso con la compilación de la nueva hoja de problemas para obtener el PDF, que se publicaba en el aula virtual o se repartía impresa en papel.

Con este planteamiento cada estudiante sólo recibe una selección de problemas. Nuestro cliente se planteaba la posibilidad de ofrecer el acceso al conjunto completo de ejercicios, de forma que cualquier persona se pueda confeccionar sus propias hojas de problemas resueltos y así impulsar el autoaprendizaje.

3. Alcance, objetivos y licencia

El objetivo general del proyecto es el desarrollo o adaptación de una solución web para la presentación y búsqueda de problemas resueltos y para la construcción de hojas de problemas con sus posibles soluciones. La solución web debe aceptar y almacenar problemas con varias soluciones escritos en LaTeX y debe producir documentos PDF con hojas de problemas seleccionados y sus soluciones.

Destacamos algunos requisitos generales del producto a elaborar:

- Los problemas se almacenarán en formato LaTeX, además de su posibles imágenes asociadas, pero se previsualizarán con la tipografía aplicada, por ejemplo en PDF.
- Los problemas se podrán etiquetar según el tipo de problema. Estas etiquetas servirán para dirigir las búsquedas.
- Los problemas se podrán seleccionar e incorporarse a una hoja de ejercicios.
- Se podrá generar un documento PDF a partir de los problemas seleccionados.
- Es muy importante obtener una versión funcional e implantada en servidores de la universidad, así que la interfaz inicial será sencilla.
- Habrá dos tipos de usuario: profesores y estudiantes.

Respecto a la **licencia**, habrá que realizar un producto de código abierto, que se compartirá mediante GitHub con una licencia GNU GPLv3, que da la libertad de usar, estudiar, compartir (copiar) y modificar el software manteniendo la misma licencia. Las características más destacables de la licencia (véase https://choosealicense.com/licenses/gpl-3.0/) son:

1. **Permiso** para uso comercial, distribución, modificación, uso de patente y uso privado.
2. **Condiciones:** revelar el código fuente, incluir la licencia, compartir con la misma licencia y documentar cambios.
3. **Limitaciones** de responsabilidad y garantía.

Sin embargo no se desean compartir los contenidos de la BD (base de datos) por otros medios diferentes a la consulta y extracción de hojas de problemas mediante el producto de este proyecto.

4. Gestión

4.1. Alcance (requisitos)

En el anexo C se listan los requisitos y exclusiones aceptadas para la realización del producto. La figura 1 muestra la EDT o descomposición en bloques de trabajo del TFG. El paquete TFG 3 Otros, contiene las actividades de puesta en producción del producto desarrollado. Entre los principales entregables se encuentra el análisis y diseño del producto y su implementación.

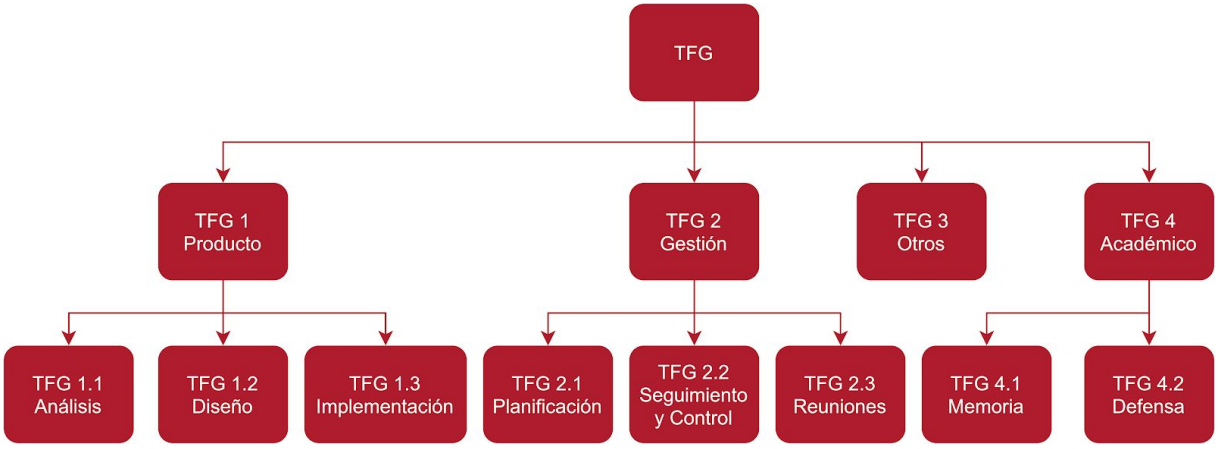

Figura 1. EDT del proyecto.

4.2. Cronograma

La figura 2 incluye los principales hitos del proyecto y la figura 3 el cronograma de actividades, distribuidas por semana, correspondientes a la EDT de la figura 1.

Hito	Febrero				Marzo				Abril				Mayo					Junio					Julio	
	S01	S02	S03	S04	S05	S06	S07	S08	S09	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	S21	S22	S23
Inicio TFG							◆																
Base de datos								◆															
API REST: Persistencia									◆														
API REST: Compilación LaTeX											◆												
Web: Usuario														◆									
Web: Admin																	◆						
Puesta en producción																			◆				
Corrección errores																			◆				
Académicos																							
Memoria revisar																				◆			
Depósito																					◆		
Defensa TFG																							◆

Figura 2. Diagrama de hitos principales del proyecto.

Periodo ejecución paquete TFG	Febrero				Marzo				Abril				Mayo					Junio				Julio	
	S01	S02	S03	S04	S05	S06	S07	S08	S09	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	S21	S22	S23
1.1. Análisis																							
1.2. Diseño																							
1.3. Implementación																							
2.1. Planificación																							
2.2. Seguimiento y control																							
2.3. Reuniones																							
3. Otros																							
4.1. Memoria																							
4.2. Defensa																							

Figura 3. Cronograma del proyecto.

4.3. Recursos (dedicaciones)

En la figura 4 se muestran las horas semanales dedicadas a cada uno de los paquetes de trabajo de la EDT del proyecto (en cada paquete de trabajo se obtiene un entregable).

Dedicación en horas paquete TFG	Febrero				Marzo				Abril				Mayo					Junio				TOTAL
	S01	S02	S03	S04	S05	S06	S07	S08	S09	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	S21	
1.1. Análisis							10	1	1	1	1	1	1	1	1	1	1					20
1.2. Diseño								10	2	1	1	1	1	1	1	1	1					20
1.3. Implementación								10	20	20	20	20	20	20	20	20	10					180
2.1. Planificación							10	1	1	1	1	1	1	1	1	1	1					20
2.2. Seguimiento y control								1	1	1	1		1	1	1	1				1	1	10
2.3. Reuniones							1	1				1					1		1	1	1	7
3. Otros																2	2	3	10	10		27
4.1. Memoria							1	1	1	1	1	1	1	1	1	1	1	5	10	20	10	56
4.2. Defensa																						
Total							22	25	26	25	25	25	25	25	25	27	17	8	21	32	12	340

Figura 4. Horas de dedicación a cada paquete por semana redondeadas a horas enteras.

5. Análisis y diseño

5.1. Estudio de alternativas de código abierto

Se empezó buscar soluciones adaptables de código abierto que permitieran lograr una solución más elaborada. La web Overleaf tenía un repositorio público y de código abierto en GitHub que permite almacenar, modificar y compilar documentos LaTeX en la nube. Se podría reutilizar el módulo de gestión y almacenamiento de ficheros en la BD, junto con el proceso de compilación.

Tras desplegar el proyecto con Docker y estudiarlo con detenimiento, la pregunta era si convenía realizar el proyecto a partir de Overleaf. Se observó lo siguiente:

- Ventajas: La capa de persistencia abarcaba más de lo necesario, en particular el almacenamiento de ficheros LaTeX junto con las imágenes que pudiese contener. Parte de la capa de lógica estaba también implementada.

- Inconvenientes: Es un producto de una dimensión muy grande como para poder comprender a fondo su funcionamiento durante el desarrollo del TFG. Utiliza tecnologías complejas que precisarían de formación incluyendo NodeJS, MongoDB, Redis y CoffeeScript.

Se concluyó que debido al largo periodo de formación necesario no se lograría desplegar una versión funcional en este TFG. Por tanto se optó por desarrollar una solución menos ambiciosa desde cero que ofreciera una funcionalidad básica pero suficientemente interesante.

5.2. Inicio del proyecto desde cero

Una vez capturados los requisitos de la tabla C1 (ver apéndice C) identificamos tres retos a superar para conseguir resolver el problema al que nos enfrentamos:

- **Almacenamiento de imágenes:** los archivos LaTeX pueden contener instrucciones en las cuales se indica que se debe de introducir una imagen, ya sea en formato PDF, JPG o PNG. Hay dos posibilidades, almacenar las imágenes dentro de la BD, tipo BLOB, o guardar referencias a los ficheros con las imágenes en las filas de la BD y almacenar dichas imágenes en el sistema de archivos (*Filesystem*). Se optó por la segunda opción, por dos razones, en primer lugar porque las BBDD con elementos de tipo BLOB tienden a ser más ineficientes, y en segundo lugar porque es necesario almacenar las imágenes para compilar el fichero correctamente.
- **Compilación de ficheros LaTeX:** antes de almacenar en la BD un fichero con extensión *.tex*, será necesario compilarlo para comprobar que es un fichero válido. Además, el programa deberá generar nuevos ficheros LaTeX para contener las hojas de problemas, y por tanto necesitamos conocer y analizar las opciones disponibles. Inicialmente se utilizó un compilador desplegado como Docker, pero más adelante se observó las ventajas de utilizar el compilador **pdflatex** desde la línea de comandos.
- **Procesamiento de lenguaje LaTeX en la web:** nuestro cliente, además de descargar el PDF de cada problema, necesita que los problemas se presenten de forma legible en la web. Es decir, necesitamos alguna herramienta que compile LaTeX en el lado del cliente y permita presentar el resultado. La mayoría de páginas web consultadas utilizan **MathJax**, una librería de JavaScript que permite la visualización de código LaTeX en el navegador.

Además decidí utilizar las mismas herramientas que ya había aprendido durante mis prácticas curriculares en la empresa Arsys:

- Desarrollo **API/REST** con **Python y Flask**.
- Desarrollo de un **servicio web** con **PHP**.

5.3. Arquitectura

Sigue el conocido patrón MVC (modelo-vista-controlador), estándar de desarrollo de sitios web. El modelo reside en la API/REST, implementada en Python, que es llamada por el controlador, el cual transfiere los datos a la vista. Los dos últimos se implementaron en PHP.

Las tres capas se han implantado en dos máquinas según se muestra en la figura 6. Las capas de persistencia y lógica de negocio (modelo) residen en una máquina. La capa con la interfaz de usuario se ha desplegado en una máquina diferente. Se ha tratado de mantener

la coherencia en las pruebas y producción de los módulos de la lógica y la interfaz para minimizar los errores.

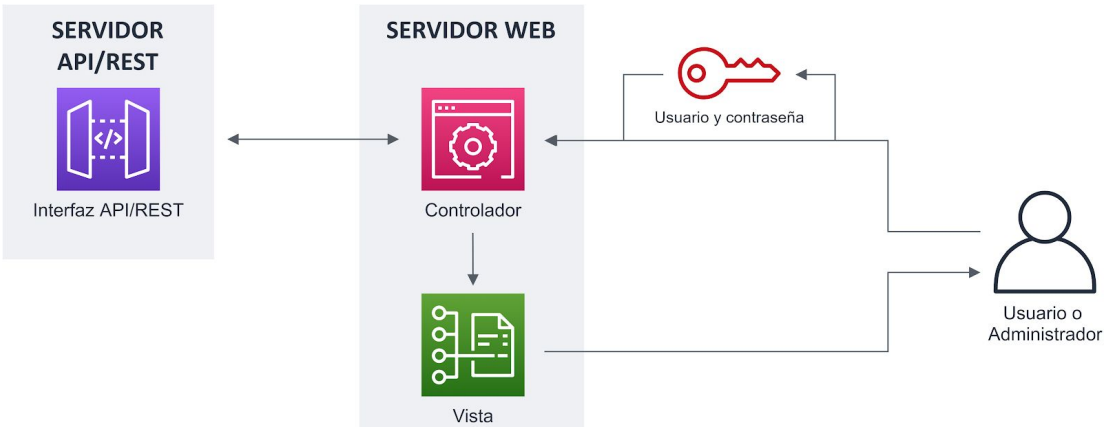

Figura 6. Diagrama de funcionamiento conjunto.

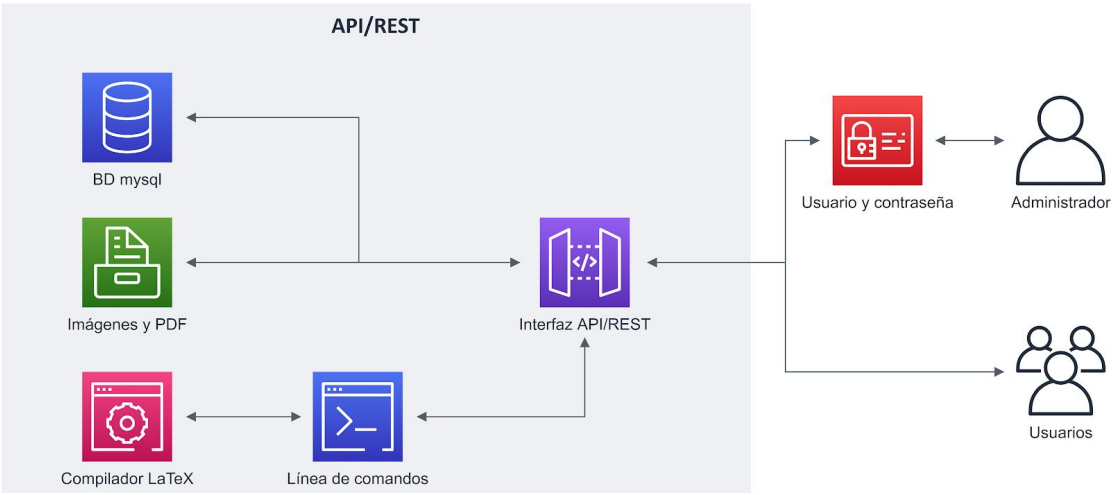

Figura 7. Diagrama funcionamiento API/REST.

A continuación explicamos el proceso de implantación de los módulos de esta arquitectura:

- **Servidor API/REST**. Se desplegó en una máquina virtual con sistema operativo Ubuntu 18.04 Server. Además, se configuró un adaptador de red para acceder al host mediante SSH (Secure Shell).
  En esta máquina se instaló y configuró un servidor de bases de datos MySQL. Se eligió MySQL por ser un sistema ya conocido, gratuito y muy extendido. Se creó la BD **problemath** (figura 5) con los usuarios *user* (acceso de lectura), y *admin* (todos los permisos).
  Antes de crear la API/REST se creó un entorno virtual con el módulo de Python **venv**. Un entorno virtual permite abstraernos del host donde está alojada la aplicación lo que facilitará el paso de un ambiente de pruebas a uno de producción. Otra ventaja de este enfoque es que podemos generar un fichero de configuración en el que podremos guardar las dependencias de nuestro entorno, facilitando así, otra vez, un futuro paso a producción.
  Se instaló la distribución de software LaTeX, llamada **texlive**. Esta distribución incluye varios compiladores, en particular **pdflatex**, que será el que usemos, y una serie de paquetes necesarios en tiempo de compilación. Por último se instaló un descompresor de ficheros .zip. La figura 7 muestra cómo se comunican las partes de este módulo.

- **Servidor WEB.** Se desplegó con la misma configuración que el anterior. Tras desplegar la máquina virtual se instaló el servidor web Apache y PHP. Una vez instalados los programas se configuraron un cortafuegos y los sitios virtuales (Virtual Hosts) que permiten encapsular detalles de configuración, además de alojar más de un dominio en el servidor. Esto, a priori puede parecer innecesario ya que inicialmente habrá un solo dominio, pero es una buena práctica para aislar las opciones de configuración.

5.4. Funcionalidad principal

La mayor parte de la funcionalidad la se ofrece desde la **API/REST** y se distinguen las opciones de libre acceso y las que precisan contraseña. Las de libre acceso son:

- **Búsqueda de problemas:** se podrán realizar búsquedas en base a palabras clave, revista y proponente del problema. Las palabras clave tienen relación con los distintos campos de las matemáticas. El resultado será una lista en formato JSON conteniendo la información básica de cada problema.
- **Detalle de un problema:** se podrá obtener más información de un problema, accediendo a sus soluciones, si contiene o no imágenes, etc.
- **Obtener PDF de un problema:** se puede obtener un problema de manera individual en formato PDF. Este archivo se puede generar con o sin soluciones.
- **Generar hoja de problemas:** se puede obtener una hoja de problemas en formato PDF a partir de una lista de problemas. En este hoja se podrán incluir o no las soluciones de cada uno de los problemas.

Las funcionalidades que exigen iniciar sesión de administrador son las siguientes:

- **Introducir/borrar un problema:** un administrador podrá introducir un nuevo problema con soluciones a la aplicación. Los formatos de archivos permitidos son *.tex* y *.zip.* El formato *.zip* se usa exclusivamente para adjuntar imágenes al enunciado/solución.
- **Crear administrador:** podrá crear nuevos administradores.
- **Cambio de contraseña:** un administrador podrá cambiar su contraseña.

Todas la funcionalidad de la API/REST está documentada en el siguiente enlace: https://stoplight.io/p/docs/gh/mahillo97/problemathapi?group=master&utm_campaign=publish_dialog&utm_source=studio

El módulo WEB utiliza las funcionalidades de la API/REST. Sin embargo, hay una funcionalidad muy importante del proyecto que es la **visualización de LaTeX en la web**. Para ello se utilizó la librería **MathJax** de JavaScript que trata texto LaTeX sin compilar.

5.5. Diseño de la interfaz

El anexo D muestra imágenes de interfaz web. Las siguientes páginas son de acceso público:

- **Página de inicio/buscador:** permite realizar las búsquedas por etiquetas.
- **Resultado de búsqueda:** ofrece una lista paginada de problemas mostrando su información principal. Desde esta página es posible añadir problemas a nuestra hoja de ejercicios.

- **Información de problema:** visión más completa de la información de un problema. Además permite ver las soluciones y descargar el problema en formato PDF de los dos modos vistos anteriormente.
- **Hoja de problemas:** visualización de los problemas seleccionados hasta el momento. Permite ordenarlos y seleccionar las soluciones que se deseen y guardar estos cambios. Por último, una vez confeccionada podemos descargar la hoja de problemas en formato PDF.
- **Página de error:** se incluye una página de error cuando se produzcan fallos con la aplicación.
- **Inicio de sesión:** se incluye una página para iniciar sesión como administrador.

Las siguientes páginas se acceden tras iniciar sesión de administrador:

- **Dashboard:** panel de control para administradores. En un futuro se espera ampliar sus funcionalidades.
- **Subir un problema:** permite rellenar los campos correspondientes y subir un problema a la BD.
- **Borrar un problema:** da la posibilidad de borrar un problema introduciendo su identificador.
- **Añadir administrador:** permite crear un nuevo administrador en la BD.
- **Cambiar contraseña:** modifica la contraseña del administrador actualmente registrado.

5.6. Diseño de datos

Ya se ha explicado en la sección 5.2 que se decidió almacenar las imágenes en archivos del sistema de ficheros en lugar de usar campos de tipo BLOB. La figura 5 contiene el esquema de la BD.

La tabla central de la BD es *problem*. Su campo Tex almacena el enunciado del problema en lenguaje tex. URL_PDF_State y URL_PDF_Full contienen rutas a los ficheros PDF con el enunciado y con el enunciado y soluciones, respectivamente. La tabla *dependency* implementa una relación M:N y su campo URL contiene referencias a ficheros de imágenes asociadas a enunciados (PDF, png,...). *Solution* contiene soluciones a problemas que se almacenan en su campo Tex, también en formato tex. Los campos Dep_State y Dep_Solu de las tablas *problem* y *solution* indican si enunciado y solución tienen alguna relación con la tabla de *dependency*. Las tablas *tag* y *package* son nombres que se asocian a los problemas mediante sendas relaciones M:N con la tabla *problem*. La tabla de usuarios contiene a los administradores del proyecto donde el campo *password* contiene el hash (SHA256) de la contraseña.

Los archivos PDF generados por la aplicación y las imágenes de los enunciados, se guardan en el sistema de ficheros del host que aloja la aplicación. Tenemos una carpeta Data que contiene un directorio para cada problema, en esta se almacenan los 2 PDF del problema. Hay otra carpeta llamada tmp que almacena ficheros temporales, por ejemplo al introducir a la aplicación un nuevo problema. También hay otro directorio llamado dp, donde se almacenan todas las dependencias de los problemas (imágenes, archivos PDF, etc).

Por último, las respuestas de la API/REST se envían mediante tres tipos de objeto JSON. La figura 8 muestra un ejemplo de tipo problema base, la figuras 9 un ejemplo de tipo problema completo y la figura 10 un ejemplo de tipo solución.

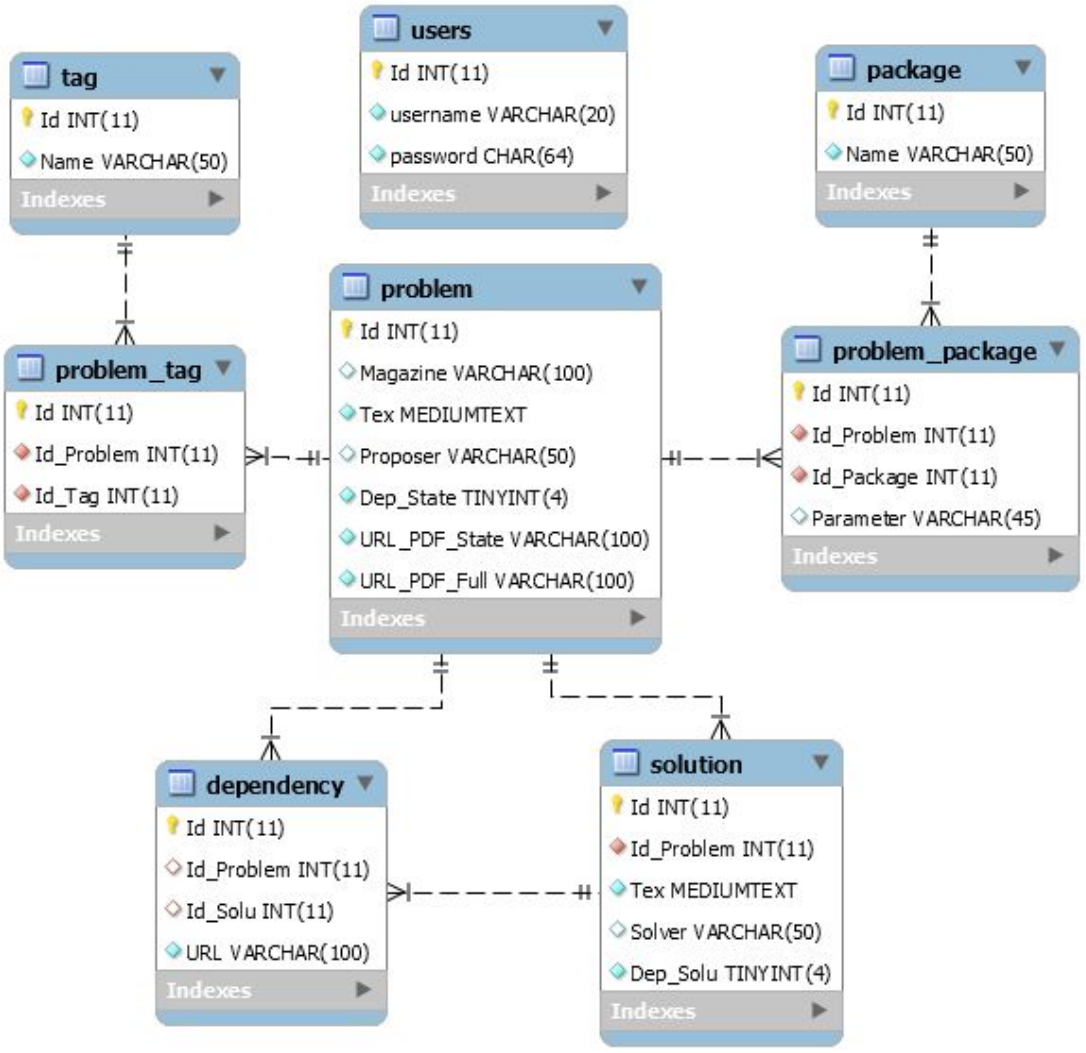

Figura 5. Esquema relacional con añadidos de la BD.

```
{
    "id": 12,
    "magazine": "The American Mathematical Monthly",
    "proposer": "Julien Sorel",
    "tags": ["combinatoria"],
    "tex": "Tex del enunciado del problema"
}
```

Figura 8. Tipo de dato Problema Base

```
{
    "dep_state": 0,
    "id": 12,
    "magazine": "The American Mathematical Monthly",
    "proposer": "Julien Sorel",
    "solutions": [
        {
            "id": 12,
            "solver": "Alejandro Mahillo",
            "tex": "Tex de la Solución"
        }
    ],
    "tags": ["combinatoria"],
    "tex": "Tex del enunciado del problema"
}
```

Figura 9. Tipo de dato Problema Completo

```
{
    "id": 12,
    "solver": "Alejandro Mahillo",
    "tex": "Tex de la Solución"
}
```

Figura 10. Tipo de dato Solución

6. Implementación

En este apartado revisamos las cuestiones tecnológicas más destacables junto a los principales problemas superados. Los anexos E, F y G contienen ejemplos de documentos PDF generados con la aplicación.

6.1. Tecnologías y herramientas

Durante el desarrollo de este TFG se han utilizado las siguientes herramientas de desarrollo de aplicaciones de software:

- **Máquinas virtuales**: se utilizó **VirtualBox** para emular el futuro entorno de producción.
- **Entorno de desarrollo:** se utilizó **Visual Studio Code** tanto para el desarrollo de la API/REST y del servidor web. Este entorno permite establecer conexiones SSH con máquinas virtuales, y escribir instrucciones en una terminal. Para la API/REST también se utilizó **Postman** que permite enviar peticiones web para realizar las pruebas.
- **Control de versiones:** se utilizado **Git** y el repositorio de **GitHub**. El código es accesible en un repositorio público de GitHub.
- **Documentación:** para crear la documentación de la API/REST se utilizaron las herramientas proporcionadas por **Stoplight**.

Para la programación de los módulos se ha utilizado:

- **Módulo API/REST:** el framework de Python **Flask** y en particular la extensión **Flask-RESTful**.
- **Módulo WEB:** se programó en PHP y para cuestiones gráficas se usaron las librerías de CSS y JS **Bootstrap 4.4.1**, **JQuery 3.4.1**, **Font Awesome 5**, **MathJax 3**. También se ha utilizado **TikzJax** para la visualización de dibujos creados con LaTeX.

Por último se utilizó el servidor HTTP **GUnicorn** para el módulo de la API/REST por sus sinergias con Flask y un servidor **Apache** para el módulo web**.**

6.2. Compilación e introducción de nuevos problemas

En el proceso de introducción de problemas había algunos puntos críticos que superar:

- **Momento de la compilación**: nuestra aplicación debía devolver dos documentos PDF, uno con el enunciado y otro con las soluciones. Había dos opciones, generar los PDF cuando fuesen solicitados o generarlos en el momento de introducir nuevos problemas y almacenados. La compilación exige cierto tiempo. Se decidió almacenar los PDF aunque se utilice mayor espacio de disco.

- **Preámbulo de los documentos *.tex*:** el preámbulo de un fichero LaTeX es la zona donde se declaran los paquetes, se definen nuevos comandos y estilos y se escriben las directrices a tener en cuenta en la compilación. Es una zona muy conflictiva cuando se unen varios documentos LaTeX ya que podrían contener instrucciones contradictorias o definir instrucciones con distintas funciones.
- **Composición de la hoja de problemas:** una funcionalidad muy importante es la generación de una hoja de problemas en formato PDF. Como hemos visto en el apartado anterior, varios problemas en tiempo de compilación han sido eliminados tratando el preámbulo de forma correcta, pero eso no implica que en el cuerpo del documento no nos podamos encontrar con otro tipo de incompatibilidades. En particular, tiene que ver con la inclusión de imágenes. Se debe de incluir la ruta de una imagen, pero es posible que en 2 documentos distintos haya fotos con el mismo nombre, por ello es un punto sensible que habrá que tener en cuenta.
- **Tratamiento de imágenes:** los problemas pueden contener imágenes, por tanto debemos aceptar en la subida de archivos tanto ficheros de texto plano como ficheros *.zip* con las imágenes. Estas imágenes se pueden incluir en el LaTeX en formatos como PDF, .png, .jpg, u otros. Pero los tres mencionados son los más comunes. Mostrar en el navegador una imagen PDF no es una tarea sencilla. Por ello se han transforman antes a otro formato de imagen.

La **solución** a estos problemas fue introducir los problemas de la siguiente forma:

1. Se sube un fichero con el enunciado y de 1 a 10 ficheros que contienen las distintas soluciones. El fichero se envía a la API/REST como *.tex* o *.zip,* cuando el fichero LaTeX incluye figuras.
2. Se almacenaban los ficheros en una carpeta de ficheros temporales y si el fichero tiene extensión *.zip* se descomprimía.
3. Se leían los ficheros *.tex* buscando, con expresiones regulares, donde había figuras. Una vez encontradas, las referencias se reemplazaban por identificadores únicos aportados por la BD cuando se introducían en esta.
4. Además, si la imagen venía en PDF, se utilizaba el conversor **pdf2svg** para convertirla a *.svg*. Se probaron otras alternativas como **ImageMagick** y **pdftoppm** para pasarlo a *.png,* pero se descartaron. Nótese que el fichero *.svg* se utilizará solo para la visualización web, mientras que en la compilación se utilizará el PDF.
5. Se guardan el resto de datos en la BD: etiquetas, problema y solución.
6. Una vez tenemos los datos correctos procedemos a compilar los ficheros necesarios, es decir, decidimos esta alternativa puesto que es en el proceso de subida donde informamos de si hay errores y ofrecemos un servicio más rápido a los usuarios. Por ello, se compilan los ficheros PDF, generamos dos, uno con soluciones y otro sin soluciones. Para ello generamos dos ficheros *.tex* auxiliares donde escribimos el texto de los ficheros de subida tras ser tratados, uno sin soluciones y otro con ellas. Después compilamos dichos ficheros con **TexLive** que nos permite compilar por mandatos de consola.
7. En caso de éxito se guardan los PDFs en una carpeta con el identificador del problema. En caso de error no se almacena nada. En ambos casos, los ficheros auxiliares se borran, a excepción de las fotos cuando ha compilación sin errores.

Además, para solucionar los problemas del preámbulo, debemos solicitar al cliente simplificar este apartado para facilitar la compilación, sólo se incluirán los paquetes necesarios. Se limitarán ciertos paquetes incompatibles, por ejemplo, solo se podrá incluir un paquete de tipo de idioma y codificación para evitar problemas, en particular deberá

utilizar la codificación UTF-8 y el paquete de idioma, si se quiere incluir, deberá ser el de español.

6.3. Almacenamiento de la hoja de problemas

La API/REST ofrece la posibilidad de generar una hoja de problemas cuando le pasamos una lista de identificadores de problemas. Además, se pueden incluir también los id de las soluciones para que aparezcan junto con el enunciado. Esta tarea presenta 2 problemas:

- Mismas consultas generan el mismo fichero aunque ya exista en el servidor la respuesta de esta.
- El compilador debe almacenar el fichero en disco. Lo ideal sería poder compilar la hoja de problemas como PDF y enviarla directamente desde memoria, sin almacenar el fichero previo en el sistema de archivos.

Ante esta problemática se presentaron las siguientes alternativas:

- Usar una cookie de sesión en la API/REST. Así, cuando se realiza la misma petición podemos devolver el PDF disponible y, si es distinta, se borra ese PDF y se crea el solicitado. Aun así, esto nos sigue obligando a borrar hojas cada cierto tiempo, cuando expire la cookie de sesión.
- Cachear las peticiones PDF de forma que si una petición ya ha sido realizada, esta no genere el PDF de nuevo y devuelva uno ya existente. Se podría generar en la BD una tabla que relacione peticiones con PDFs.
- No realizar ningún tratamiento con las peticiones, pero generar un proceso que se ejecute cada 30 minutos para borrar los PDFs generados.

Se optó por la última **solución** por su simplicidad. La primera se descartó porque usar una cookie de sesión no mejora gran cosa los resultados de la solución implementada y era más laboriosa. La segunda se descartó porque es bastante improbable que varios usuarios generen la misma hoja de problemas.

6.4. Búsqueda de problemas desde la web

Inicialmente se diseñó una búsqueda simple con un campo donde introducir palabras claves por las que realizar la búsqueda en la BD. Esto producía los siguientes problemas:

- Las palabras del campo de búsqueda debían estar separadas por comas. No se realizaba un formateo y validación de esta cadena antes de realizar la petición a la API/REST.
- La API/REST devolvía todos los resultados, pero en la web se muestran 5 por página. Este enfoque era ineficiente, se devolvían más resultados que los necesarios.

Las **soluciones** a estos problemas fueron las siguientes:

- Ahora, cada vez que se introduce una palabra clave en el buscador y se teclea una coma o se pulsa *Intro,* la etiqueta se almacena debajo del campo de texto permitiendo eliminarla o seguir añadiendo distintas etiquetas. De esta forma mejoramos el formateo de parámetros y se simplifica la interfaz para el usuario.
- El segundo problema se solucionó añadiendo dos parámetros a las consultas en la API/REST: un índice y una cantidad. La API devolvería esa cantidad de resultados a partir del índice, mejorando así el rendimiento.

6.5. Petición de hoja de problemas desde la web

La API/REST ofrece la posibilidad de generar una hoja de problemas, para ello le pasamos una lista de identificadores en orden y en función de ese orden generamos la hoja de problemas. En un principio en la web solo daba la posibilidad de añadir un problema con soluciones o no, sin importar el orden.

La **solución** fue utilizar la funcionalidad **Sortable** de **JQuery**. Esta presenta listas de elementos y permite reordenarlos arrastrando y soltando. Es necesario fijar el orden deseado pulsando un botón. De esta forma conseguimos una interfaz bastante intuitiva para la composición de hojas de problemas.

6.6. Visualización de LaTeX en el navegador

La herramienta utilizada para realizar esta tarea fue **MathJax**, una librería de JavaScript que permitía la visualización de las instrucciones básicas en los navegadores. Hubo dos problemas fundamentales utilizando esta librería.

- Ciertos comandos básicos de LaTeX no los mostraba correctamente, así como las referencias a ecuaciones.
- Las imágenes no se veían puesto que buscaba en la ruta local de mi servidor web y no en la API/REST.

La **soluciones** a estos problemas fueron las siguientes:

- Para la correcta configuración web tuve que leer la documentación de MathJax y encontrar una configuración que se adaptara a mis necesidades. La configuración utilizada se puede ver en la figura 11.

```
window.MathJax = {
    loader: {
        load: ['[tex]/ams']
    },
    tex: {
        packages: {
            '[+]': ['ams']
        },
        tags: 'ams',
        inlineMath: [['$', '$'],['\\(', '\\)']]
    }
};
```

Figura 11. Configuración MathJax

- Para poder ver correctamente las imágenes tuve que modificar los entornos que aparecían en el código LaTeX por etiquetas de imagen de HTML. Además de modificar las rutas relativas a la API/REST con peticiones a la API/REST por la imagen que necesitamos mostrar.

6.7. Visualización de figuras TikZ en el navegador

Una vez resuelto el problema de visualización de LaTeX en el navegador, surgió otra dificultad: los dibujos generados directamente con LaTeX no se mostraban correctamente. Estos dibujos se generan con TikZ, una herramienta de macros de alto nivel que utiliza otra

de más bajo nivel para la producción de dibujos vectoriales. Estos dibujos se pueden incluir en LaTeX con una serie de comandos, que MathJax no es capaz de comprender. Como MathJax no ofrecía soluciones al problema se buscaron otras alternativas.

La **solución** fue utilizar **TikzJax**, una herramienta que analiza y compila en el navegador este tipo de dibujos. No trabaja directamente sobre un fichero tex, es necesario incluir etiquetas de script en html cada vez que aparece una imagen de este tipo. Por ello, además de incluir estas librerías, tuve que modificar el texto del problema con JavaScript para poder incluir correctamente las etiquetas de scripting, esta modificación del texto se hizo de manera análoga a cuando tuve que cambiar los entornos de imágenes por etiquetas de HTML.

7. Implantación

Uno de los requisitos iniciales era que el producto quedara desplegado en un servidor de la Universidad. Finalizado el periodo de desarrollo, me puse en contacto con el servicio informático para este fin. Me pidieron que hiciera el despliegue en un único servidor. Sólo podían proporcionarme un segundo servidor si era estrictamente necesario. Realicé el despliegue conectándome mediante VPN. El cambio de dos servidores a uno supuso realizar algunos cambios muy simples. Pero podían surgir problemas en las llamadas a la API/REST. Para solucionarlo cambié la IP por *localhost.* Los pasos del despliegue se han explicado en la sección 5.3 (arquitectura) y 6.1 (tecnologías). Después, cloné los proyectos de Github para desplegarlos en el nuevo servidor.

7.1. Despliegue de la API/REST

Puesto que el mini framework **Flask** utilizado durante el desarrollo, está pensado para un entorno de pruebas, busqué una alternativa orientada a producción. La documentación de **Flask** recomendaba varios servidores. Decidí utilizar **GUnicorn** por las facilidades de puesta en producción. GUnicorn se lanza desde la consola (línea de comandos), y se especifica la aplicación a desplegar. Por tanto, es muy simple de utilizar pero poco práctico ya que en caso de surgir problemas exige realizar de nuevo la misma operación desde la consola.

Para resolver este problema creé un nuevo servicio para el despliegue de la aplicación. Así, pude programar el servicio para forzar su inicio y las demás opciones de configuración del servicio son útiles para que el servicio esté siempre activo.

7.2. Despliegue en el servidor WEB

Era la primera vez que configuraba de un servidor web, así que seguí los pasos que me indicaron desde el servicio informático:

1. **Protocolo HTTPS.** Me proporcionaron tres ficheros, dos de ellos con extensión *pem.* El primero contenía el certificado del servidor y el segundo la cadena de certificados para la clave pública. El tercero era la clave privada de extensión *.key*.

   Estos certificados eran utilizados para la conexión por el puerto 443, es decir, la conexión HTTPS, por último debía redirigir el tráfico que llegase por el puerto 80 al puerto 443.

2. **Securización.** Me proporcionaron también algunos test de seguridad que debía superar. Eran SSLLabs del servicio de Qualys y se pueden realizar desde la siguiente página web: https://www.ssllabs.com/ssltest/. Inicialmente calificó el producto con B (Figura 12) pero introduciendo algunos cambios conseguí calificación A (Figura 13). Para realizar estas mejoras seguí las recomendaciones de la documentación proporcionada por SSLLabs junto con Qualys. Las modificaciones afectaban únicamente al fichero de configuración del servidor web Apache. Apache simplifica mucho esta tarea, que inicialmente parecía tan compleja.

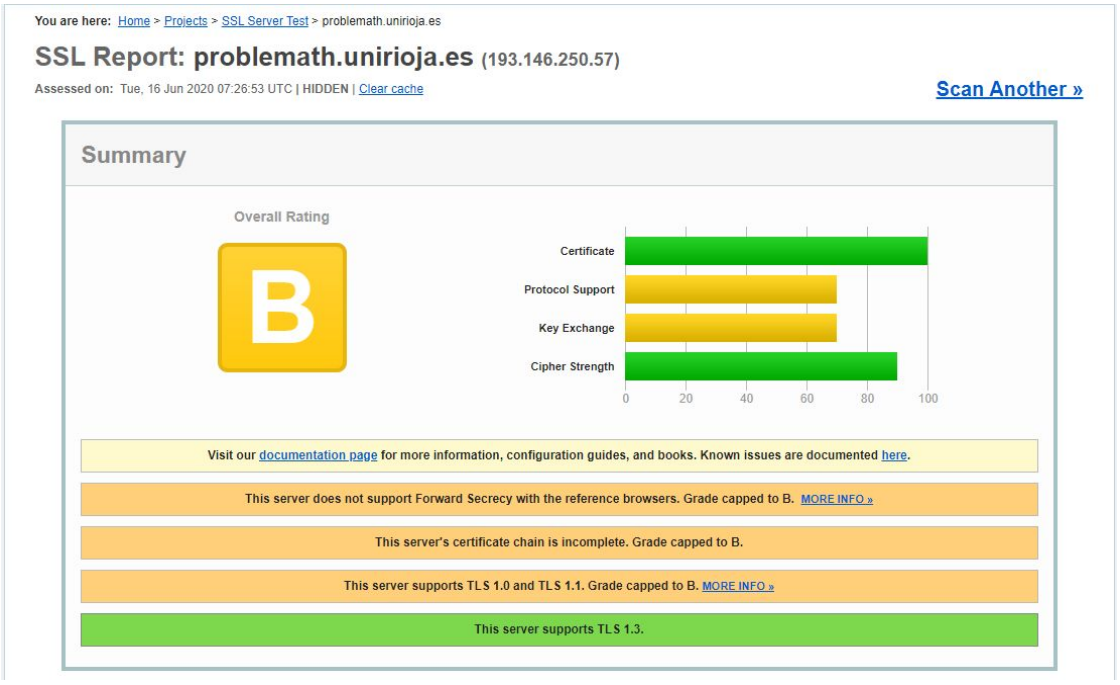

Figura 12. Calificación inicial servidor web.

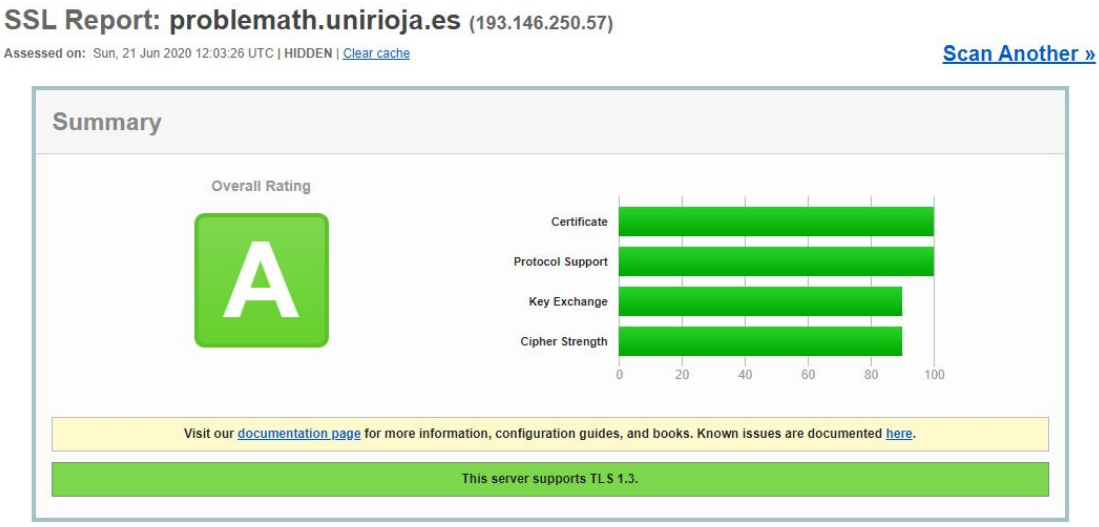

Figura 13. Calificación final servidor web.

7.3. Validación con el cliente

Una vez finalizado el despliegue del proyecto el cliente comenzó a probar las funcionalidades implementadas, el resultado en general fue satisfactorio, excepto la visualización de LaTeX en el navegador. Encontramos los siguientes problemas:

- Había entornos de LaTeX (instrucciones entre los comandos \begin{} y \end{}) que no se ejecutaban correctamente.
- Las instrucciones de los paquetes de español, como por ejemplo \sen para indicar que queremos escribir la función seno, no se visualizaban correctamente.
- Las imágenes con pies de página no se veían. Esto era lógico puesto que MathJax no trataba las imágenes y era yo quien lo hacía y ese caso no lo había contemplado.

Estos problemas eran críticos a una semana de entregar el TFG puesto que era una funcionalidad importante para el cliente. Analicé la situación y llegamos a los siguientes resultados.

Primero busqué en la documentación de MathJax para ver si se podían incluir entornos para que la visualización fuese correcta. Por desgracia, MathJax en su documentación informa que no admite la inclusión de paquetes. Es decir, MathJax es una herramienta ineficiente para la visualización de LaTeX en la web ante problemas muy grandes. A pesar de darte la mala noticia, en la documentación de MathJax aparecen alternativas, proponían compilar el LaTeX en formato html, en vez de PDF. Buscando en internet encontré 7 alternativas distintas, pero no tenía tiempo de probar cual iba mejor en mi caso, por ello, probé directamente con uno recomendado en una página web: **make4ht,** un sistema construcción del compilador **tex4ht** el cual convierte ficheros *tex* en documentos *html* completos.

Ante esta situación teníamos los siguientes problemas:

- Este proceso de compilación se realizaba en la API/REST por tanto después este documento html debería de ser enviado al servidor web. Lo ideal sería únicamente utilizar el código que nos interesa (el del interior de los comandos \begin{document} y \end{document}). Pero esto supondría un tiempo de pruebas que no tenía.
- Otro problema eran las imágenes, teníamos problemas de compilación a la hora de generar a partir de un PDF un png, que era el proceso que seguía **make4ht.** Además las rutas generadas en el html eran relativas a la API/REST, no respecto a la web.
- Tras una primera compilación de prueba el resultado era aceptable. Pero los estilos no concordaban con el estilo de la página original.

Esto supuso el desarrollo de las siguientes **soluciones**:

- Creación de un fichero de configuración para la compilación de *tex* a *html*, esto era necesario para poder incluir las librerías de MathJax y de Bootstrap para mejorar el estilo.
- Durante el proceso de subida de problemas además de compilar los PDFs debíamos de compilar el enunciado y cada una de las soluciones pero separado para obtener el html.
- Una vez comprobé que la compilación era satisfactoria debía crear métodos que devolvieran esta información para que se pudiesen incrustar cómo iframe dentro del servidor web. Un iframe es una etiqueta html que te permite incrustar otros elementos, como PDFs, videos u otras páginas web.

- Además teníamos el problema de las imágenes, para el cual tuve que desarrollar un script dentro del fichero de compilación de LaTeX para incrustarlo en los ficheros generados.

Ahora los problemas se ven bastante mejor que antes aunque tenemos nuestra página repleta de iframes que no permiten un diseño fácil. Esta solución temporal se pretende mejorar en un futuro de una manera más eficiente.

8. Conclusiones

La realización de este TFG ha supuesto un reto en varios aspectos, pero voy a destacar tres lecciones aprendidas. La primera lección trata sobre *comprender bien las necesidades del cliente*. A pesar de ser un problema muy conocido, lo cierto es que surge en cuanto uno se pone manos a la obra. Como los datos de entrada eran documentos LaTeX, elaboré un juego de pruebas considerando un conjunto amplio de posibilidades de este lenguaje. Sin embargo, en la fase de validación final del producto, cuando se probó por primera vez el producto con documentos del cliente, se observó que el programa era incapaz de ofrecer los resultados esperados. Obviamente, no se habían considerado varios de los elementos usados habitualmente por el cliente. Por tanto, para futuros proyectos es muy importante disponer de ejemplos y datos de entrada manejados por el cliente en lugar de confiar en lo que le hemos entendido.

La segunda lección aprendida trata sobre la *calidad y manejo de la documentación*. Es muy importante disponer de documentación clara y de calidad. Cuando la documentación es compleja, se pueden buscar preguntas de usuarios sobre el problema, por ejemplo en StackOverflow. Esta situación se produjo, por ejemplo, al crear el fichero de configuración para compilar documentos tex y obtener la salida en HTML. Incluso con las respuestas de StackOverflow era muy difícil encontrar una solución al problema, debido al alto nivel de conocimientos sobre LaTeX de los participantes. Además de la calidad de la comunicación es importante cómo se maneja. MathJax tiene una buena documentación que es clara, detallada y con muchos ejemplos. Pero a veces uno se ofusca y pretende que se pueden hacer cosas que la propia documentación explica que son imposibles. Por tanto, antes de iniciar un proyecto es importante contar con documentación de calidad y prestarle la suficiente atención, para evitar riesgos y retrasos.

La tercera y última lección trata sobre la *elección de un entorno de desarrollo eficaz*. Para realizar el proyecto se utilizó Visual Studio Code. Entre sus características destaco el acceso y edición de ficheros de forma remota, por ejemplo mediante una terminal y la disponibilidad de muchos formateadores de texto, lenguajes y módulos entre sus extensiones. Manejar una herramienta integradora como esta facilita mucho la tarea, siendo esta una opción muy recomendable.

Además de las lecciones me gustaría hacer alguna reflexión sobre la realización de este proyecto. La tarea realizada me ha ayudado a crecer como profesional, observando cómo suelo afrontar los problemas y descubriendo aspectos a mejorar. En este trabajo he jugado el rol de un informático que trata de integrar las matemáticas en una solución. Sentía que tenía que hacer todo por mi cuenta, sin pedir ayuda. Sin embargo, en la fase final del proyecto he descubierto que me equivocaba. Podría haber resuelto muchos problemas de otra forma, de haber contando con la opinión de otras personas. Confrontar puntos de vista

amplía la percepción que uno tiene sobre un problema. He descubierto que los profesores están encantados de aportar ideas sobre el modo de resolver problemas. Algo que parece indiscutible es que en cualquier proyecto surgirán problemas y que conocer el punto de vista de los demás puede ser útil para evitar o superar bloqueos.

Por último, en este trabajo me ha servido para ampliar mis conocimientos sobre desarrollo web, tanto en la parte del frontend como del backend, y sobre el funcionamiento del sistema de compilación de LaTeX.